

**NETWORK PRIORITIZATION USING SDN**

**Network Programming and Application (CMPE 207)**

**Instructor: Andrew Bond**

**Fall 2019**

**Project**

**by:**

**Riya Chaugule(riya.chaugule@sjsu.edu)**

**Karan Ghorpade(karan.ghorpade@sjsu.edu)**

**Sreeparvathi Nambiar(sreeparvathi.nambiar@sjsu.edu)**

**Zaid Laffta(zaid.laffta@sjsu.edu)**

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **No.** | **Content** | **Page** |
| 1 | Abstract | 3 |
| 2 | Project Description | 4 |
| 3 | Use Cases | 5 |
| 4 | Functional Requirements | 6 |
| 5 | High Level Design | 7 |
| 6 | Implementation | 12 |
| 7 | Project Outcome and test cases | 15 |
| 8 | Future Scope | 19 |
| 9 | Individual Contribution | 20 |
| 10 | Appendix A | 22 |
| 11 | References | 23 |

1. **Abstract**

In all modern networks’ not all traffic is equal in importance, so traffic needs to be processed faster than other like video interactive session or voice sessions. Providing different priority to certain traffic over IP network is becoming an important aspect of any modern enterprise network. Not only traffic priority is important to support video and voice, it has become an essential element to support IoT which have been growing at a very fast pace during the last few years. Some applications running on IP networks are sensitive to delay as mentioned. These applications commonly use the UDP protocol as opposed to the TCP protocol. Let’s talk about the differences between UDP and TCP in a brief way, the key difference between TCP and UDP is that TCP will retransmit packets that are lost in transit while UDP does not. For a file transfer over IP network TCP should be used because if any packets are lost or arrive out of order the TCP protocol can retransmit and reorder the packets to recreate the file on the destination. That being said, UDP applications such as an IP phone, conferencing software, and video applications, any lost or delayed packet cannot be retransmitted because the voice packets come in as an ordered stream; re-transmitting packets is useless. This is the main reason that certain application needs traffic prioritization at some level to ensure high quality. In our voice call example, losing even a few packets will result in a low voice quality and jitter. Despite there are solutions like QoS, but the problem with QoS that it needs to be configured at every single device in your IP network and that can be a very large number of devices. Moreover, most of the QoS are layer 7 which means if you change your conferencing software you will need to change your QoS throughout the whole network which will have a big overhead on the IT team. SDN is a great solution for such issue. We can have pre-set policies that can be deployed according to the need of the IP network to optimize traffic forwarding. SDN will help reduce man hours and deploy any type of QoS policy in second at any giving number of devices. The idea behind this project is to utilize SDN to deploy different types of QoS policies in any giving IP network.

1. **Project Description**

Our project focuses on network packet prioritization. Instead of going to every single box in an IP network and change the QoS on it, imagine if you can do it from a centralized location. We will call it “SDN Controller” where the network engineer can go there and change different packet priority according to the need of the organization. This can be achieved by using Software Defined Network which is known as SDN. In our project the Control plane is decoupled from the data plane. The control plane in the SDN network will be in a centralized place namely the SDN controller, thus the network’s intelligence is centralized, and it can be cloud base or a server on premise depending on the IP network standards needed by a given organization. The network administrator can dynamically adjust network wide traffic flow instead of going to each and every single device within the network and change it.

In summary, our project will look at the issue of traffic flow and QoS and how we can change a network wide traffic priory without going to each and every device in the network. This can be done by using a centralized device that can be hosted in the cloud or on premise. Network prioritization using SDN will save organizations time and resources and makes it easy to prioritize certain traffic in their IP network.

A close up of a map

Description automatically generated

**Fig. 2.1: High level view of the SDN**

1. **Use Cases**

For this project we have created a custom topology with three hosts connected to two switches. We will consider these three hosts as three users. We will also consider a network administrator in this entire scenario.

**Use Case 1:** The network administrator should be able to perform IP and bandwidth allocation depending upon the network resource requirement from multiple users.

**Use Case 2:** Depending upon the requirements the network administrator should be proactively able to carry out routing and service changes.

**Use Case 3:** The functionality of deploying double QoS on multiple switches within an IP network with variables provided to the network administrator.

1. **Functional Requirements**

|  |  |
| --- | --- |
| **No.** | **Functional Requirement** |
| 1 | The users must be able to ping each other and the switches inside the network. |
| 2 | The switches must be connected to the POX controller. |
| 3 | The network administrator must be able to perform QoS operations on the links between desired source and destination. |
| 4 | The control plane and data plane must be separated while setting up the virtual network. |
| 5 | The system must provide facilities such as deletion and addition of VLANS |

1. **High Level Design**

**5.1 Functional Block Diagram:**

The Data plane and the Control plane are decoupled in the SDN Network. The Control Plane is called Logically Centralized Controller. All the information is collected by the Data Plane and are given to the Control Plane so that the administrator could make use of these information and have a general overview of the network. The administrator or the Controller then monitors the networks and make the configuration changes in the switches or the devices. Through the controller the decisions are distributed on the global view to the data plane. The user or network administrator can set the QoS and priority either manually or through a web application. The POX controller is the default controller provided inside the Mininet VM. It is the python based SDN controller. The POX controller is the brain of the entire system. The algorithm in the controller uses the queues to populate the flow table entries for Open vSwitches.

A screenshot of a cell phone

Description automatically generated

**Fig 5.1 High level design diagram**

**5.2 Logical Design of packet processing:**

The flow chart of the packet processing as it enters the switch. At first as the packet gets in the switch the switch will update the MAC address table with the source. If the packet is LLDP or bridge aggregate, it will be processed without sending it out to any ports. Otherwise, it will be processed and send out to the egress port requested by the sender of the packet.

A close up of a map

Description automatically generated

**Fig 5.2 packet processing**

**5.3 Custom Topology:**

During the initial phase of the project development the team tried to use enterprise equipment. After multiple tries with Juniper and Cisco switches that we got off eBay it turns out that it is impossible to decouple control and data plane traffic. After much research the team decided to go with Open vSwitch as it supports open flow and can be deployed using Mininet. For this project we created a custom topology consisting of 3 hosts and 2 switches.

A close up of a device

Description automatically generated

**Fig 5.3 Project topology**

**5.4 Match Tables:**

SDN uses a Match Action Table irrespective of the fact on which Layers it is functioning in. The device can act as Layer 2, Layer 3 or as a Firewall etc. The Match Action Table consists of the list of rules and priority in a table. Packet header will be in the Match Column, according to the match corresponding Action will be taken depending upon the priority.

A screenshot of a cell phone

Description automatically generated

**5.5 GUI:**

Having a centralized repository to upload automation scripts that can be deployed in SDN will be great help for any network team. This will ensure that all scripts are tested in a lab environment before it have been deployed on production network. Also, as the network team get bigger and the repository of scripts get bigger it will be easier to manage and execute changes as needed by the network admin.

A screenshot of a cell phone

Description automatically generated

**Fig 5.5 Cloud based script repository**

1. **Implementation**

We have used Mininet VM to create our topology. Mininet can be defined simply as a virtual environment acts as a test bed to implement any given software-defined networks (SDN). This environment enables SDN implementation on any device with compatible operating systems like a PC. This will allow inexpensive environment to test and do changes before deploying them in live environment. Mininet’s advantages can be summarized as the following:

* Simulate and prototyping SDN
* Virtual topology testing in a lab environment will reduce the risk of making mistakes in production environment while implementing changes
* Multiple virtual IP networks work independently on the same machine and same topology
* Cheap option to do testing without the need for real hardware

Our setup is as the following:

* GUI where network administrator can input QoS settings
* POX Controller
* Multiple OpenVSwitch
* Multiple source hosts
* A single destination host

We have created three python scripts. The topology2s3h.py will create a custom topology. The qos1.py and qos2.py scripts will create the ovs-vsctl queues on the links between the hosts. The implementation steps are as follows: (all the files can be found in our GIT repository. The link is:

<https://github.com/zaidlaffta/CMPE-207-Network-Prioritization-Using-SDN>

* Set up SSH connection with the Mininet VM from your local machine
* Start the Mininet VM
* scp all the files in the projects 'python' folder into the 'mininet' directory of your Mininet VM
* Mac users start the XQuartz once the Mininet VM is up and running
* Open three Terminal windows with XQuartz
* SSH into the mininet from all the three terminal windows
* In one of the terminal window type command 'cd Mininet'
* To start the custom topology use command 'sudo python topo2s3h.py'
* This will start the topology
* Type command 'pingall'
* The hosts won't be able to ping each other as the switches don't have the forwarding information to route the packets
* Type command 'exit' to stop the topology
* Use command 'sudo mn -c' to clear the previous emulation
* Now in the second terminal window type 'cd pox'
* Then type the following command 'python pox.py log.level --DEBUG forwarding.l2\_learning'
* This will start the l2\_learning switch
* A layer 2 switch is a type of network switch or device that works on the data link layer (OSI Layer 2) and utilizes MAC Address to determine the path through where the frames are to be forwarded. It uses hardware based switching techniques to connect and transmit data in a local area network (LAN).
* A layer 2 switch can also be referred to as a multiport bridge.
* This allows the switches to have the forwarding information about each host inside the network.
* Now follow the steps given above to start the topology again
* perform pingall and the hosts will be able to ping each other
* In the same window open the xterm for h1 h2 and h3 type 'xterm h1 h2 h3'
* In window for h1 perform 'iperf -s'
* In window for h2 perform 'iperf -c 10.0.0.1 -r'
* Observe the traffic and bandwidth
* Now we will apply the single QoS Queue.
* In the third window opened initially run the 'qos1.py' script 'sudo python qos1.py'
* This script will create single QoS with a single Queue (q0) on port s1-eth1
* This will throttle all egress traffic (going out) on that port
* Now in the same terminal window as before run the script 'sudo python qos2.py'
* Instead of applying QoS on queue 0 (default) on port s1-eth1, we will create two queues, one on which we enqueue traffic we want to pass unrestricted (at maximim bandwidth 1Gbit/s) and one on which we enqueue the traffic we want throttled to 4Mbit/s

1. **Project Outcome and test cases:**
   1. Starting the custom topology with 2 switches and 3 hosts

A screenshot of a cell phone

Description automatically generated

* 1. Starting the POX controller. The l2\_learning switch:

A close up of a newspaper

Description automatically generated

* 1. Perform iperf -s on host 1

A screenshot of a cell phone screen with text

Description automatically generated

* 1. Perform iperf -c 10.0.0.1 -r on host 2. The results are before applying the QoS which is our test case number 1.

A screenshot of a cell phone

Description automatically generated



Now we will apply QoS on the traffic going from host 2 to host 1. We will create single QoS with a single Queue (q0) on port s1-eth. This will throttle all egress traffic (going out) on that port. A significant drop in data transfer and bandwidth can be observed. This is our test case number 2.

A screenshot of text

Description automatically generated



Now instead of applying QoS on queue 0 (default) on port s1-eth1, we will create two queues, one on which we enqueue traffic we want to pass unrestricted (at maximim bandwidth 1Gbit/s) and one on which we enqueue the traffic we want throttled to 4Mbit/s. This is test case number 3

A screenshot of a cell phone

Description automatically generated

A screenshot of a newspaper

Description automatically generated

Test Case 1: After applying QoS on the entire topology i.e all the links

|  |  |
| --- | --- |
| Bandwidth before QoS | Bandwidth after QoS |
| 40.3 Gbits/sec, 43.6 Gbits/sec | 5.50 Mbits/sec, 4.23 Gbits/sec |

Test Case 2: After assigning priority to a single link between h1 and h2 and less priority to link between h1 and h3

|  |  |
| --- | --- |
| Bandwidth between h1 and h2 | Bandwidth between h1 and h3 |
| 691mbits/sec, 10.9 Gbits/sec | 415 Mbits/sec, 30.7 Gbits/sec |

1. **Future Scope**

This system can be used for multiple applications. The video streaming services can be benefitted hugely from such system. The administrator can change the bandwidth allocation for multiple users according to their requirements. Also, the IoT devices can also be connected to this system for better data transfer from the devices to the destination. For example, consider a campus network with motion sensor, camera and computers generating user traffic as shown in the topology. For a particular floor in the building, if the motion sensor detects any movement, the cameras should be given higher priority to stream HD video and web trafficssss should give lower priority. Similarly, when there is no motion, the cameras should only stream SD video and we should not let the camera's UDP traffic take over the link bandwidth.

1. **Individual Contribution**

One of the most important aspect of this project was the collaboration between our team members. We, as a group worked together to achieve all the required results. All team members worked equally in all different areas of the project. Below are the roles of all team members:

* Zaid Laffta:
  + Worked on building the whole GUI
  + Implemented different test cases using opendaylight and other SDN technology.
  + Also worked on the network part of the project, building the high-level diagrams.
  + Create the demo video
  + Wire the report
  + Debug issues with Mininet and the network topology
  + Worked on the open vSwitch
  + Created the presentation slides
* Karan Ghorpade:
  + Downloaded, Installed and Deployed Mininet environment on the local machine.
  + Created a custom pox controller to achieve the desired QoS results.
  + Researched about creation of QoS queues.
  + Created customer-oriented use cases which will help the users of this system to have an easier control over their network resources.
  + Created a step by step implementation and deployment Readme for the GitHub repository.
  + Worked in cohesion with the team on the project documentation and presentation.
* Riya Chaugule:
  + Developed python script to create custom topology for the Mininet.
  + Played a major role in creating test scenarios to carry out end to end testing of the project.
  + Worked on creating the data flow diagram for logical
  + Worked on the Mininet deployment and build 5 different labs with different number of host and switches trying to find the best implementation that will not consume the host PC resources.
  + Also worked on the slides and the report
* Sreeparvathi Nambiar:
  + worked on all test cases, Mininet deployment, and network diagrams.

In summary, all team members contributed equally in this project.

**10. Appendix A**

**Important URLs**

* GitHub URL for all source code, GUI, and .PY files

<https://github.com/zaidlaffta/CMPE-207-Network-Prioritization-Using-SDN>

* Demo video
* <https://youtu.be/m311cnKSBTE>

# **11. References**

* Opendaylight and SDN using Opendaylight : <https://www.opendaylight.org/>
* Mininet installation and operation: <http://mininet.org/download/>
* SDN use case and application: <https://www.routerfreak.com/software-defined-network-use-cases-from-the-real-world/>
* QoS basics and use cases: <https://www.networkcomputing.com/networking/basics-qos>
* Cisco SDN concepts and automation: <https://www.cisco.com/c/en/us/solutions/software-defined-networking/overview.html>
* SDN using Mininet: <http://pakiti.com/sdn-101-using-mininet-and-sdn-controllers/>
* SDN controller using Mininet emulator: <https://opensourceforu.com/2016/01/configuring-an-sdn-controller-in-open-source-mininet-emulator/>
* Switch packet flow: <https://www.geeksforgeeks.org/packet-flow-in-the-same-network/>